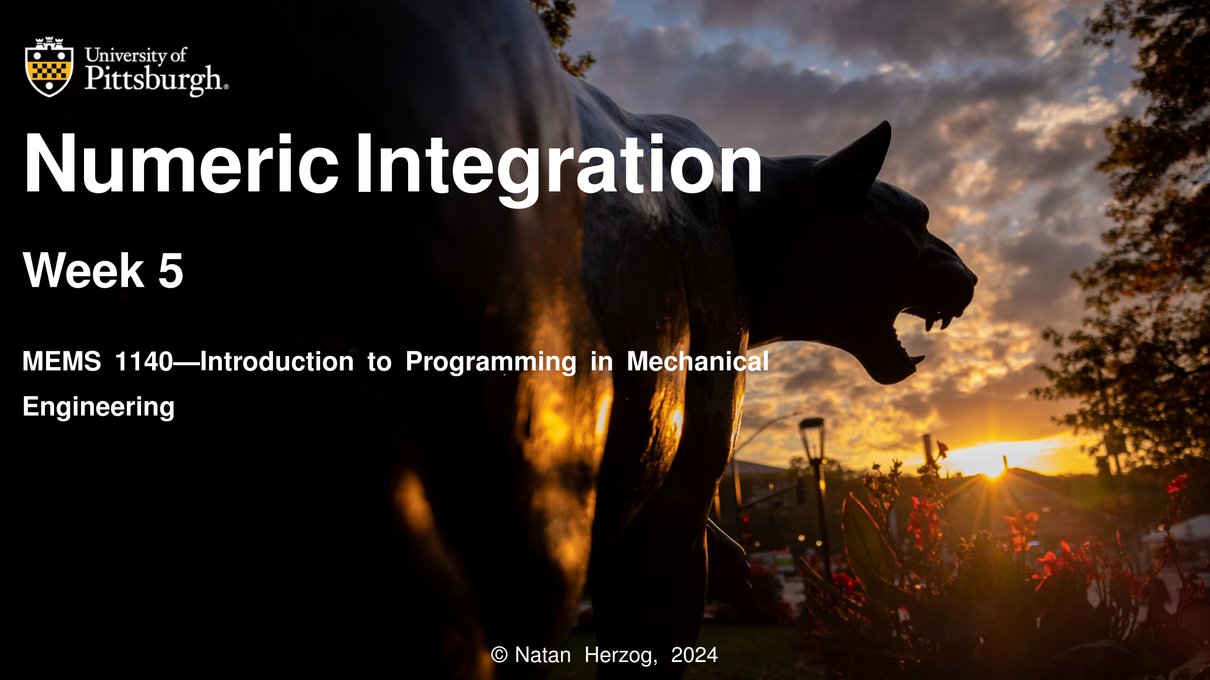


University of Pittsburgh

Numeric Integration

Week 5

MEMS 1140—Introduction to Programming in Mechanical Engineering

© Natan Herzog, 2024

Learning Objectives (L.O.)

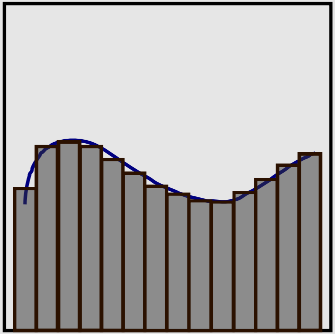
At the end of this lecture, you should understand/be able to:

- ❑ What numeric integration is;
- ❑ Why numeric integration is important on computers;
- ❑ Apply numeric integration to centroid calculations;
- ❑ How numerical results converge.

Table of Contents (ToC)

1 – Back to Introductory Calculus

Calculus class introduces integration with a picture like this:



$$A \approx \sum_{i=1}^{N} \underbrace{y(x_i)\Delta x}_{\Delta A_i}$$

Recall an actual integral:

$$A = \int_{x_1}^{x_2} \underbrace{y(x)dx}_{dA}$$

1 – Numeric Integration

As the number of rectangles (N) increases, the width (Δx) goes to 0, the sum approaches the full integral:

$$\lim_{\Delta x \to 0} \left[\sum_{i=1}^{N} y(x_i) \Delta x \right] = \int_{x_1}^{x_2} y(x) dx$$

This is the foundation of numeric integration.

Rather than doing calculus, just add up a bunch of algebra.

2 – Why Do we Care?

Mechanical engineering often requires evaluating integrals:

- Centroids $\rightarrow$ center of gravity; center of rotation; etc.
- First moments of inertia $\rightarrow$ rotational speed/energy;
- Second moments of area $\rightarrow$ bending calculations;
- Distributed loads $\rightarrow$ total load and stress profiles.

Rather than doing these by hand, it can be helpful to be able to implement numeric integration.

3 – Centroid Calculations

Recall how to calculate the centroid of an area:

$$x_C = \frac{\displaystyle\int_{y_1}^{y_2} x_{C_{dA}} dA}{\displaystyle\int_{y_1}^{y_2} dA} \qquad y_C = \frac{\displaystyle\int_{x_1}^{x_2} y_{C_{dA}} dA}{\displaystyle\int_{x_1}^{x_2} dA}$$

Where $x_{C_{dA}}$ and $y_{C_{dA}}$ are the x- and y- locations of the centroid of the dA elements.

3 – Numerical Integration of Area

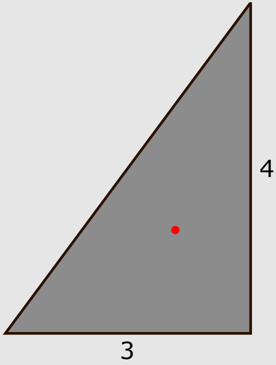
Because the integrals are nearly identical for x_C and y_C, we will only consider x_C.

Let's start with the denominator (it is simpler, after all):

$$\int_{y_1}^{y_2} dA = \int_{y_1}^{y_2} \underbrace{[x(y)]\, dy}_{dA} \approx \sum_{i=1}^{N} \underbrace{[x\,(y_i)]\,\Delta y}_{\Delta A_i}$$

3 – Analytically Evaluating Area

Consider the following triangle:

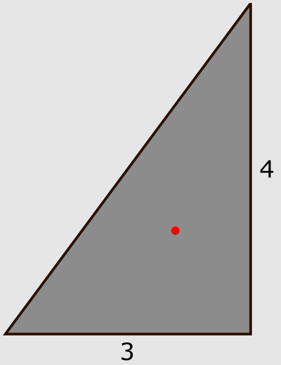


$$x(y) = 3 - \frac{3}{4}y$$

$$A = \int_0^4 x(y)dy = \int_0^4 \left(3 - \frac{3}{4}y\right) dy = 6$$

3 – Numerical Implementation

To start, define a function for $x(y)$:

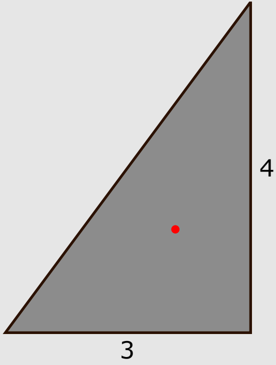



```
x = @(y) 3 - 3/4 * y;
% x(y) anonymous function
.
.
.
.
```

3 – Numerical Implementation

✓ L.O.1
✓ L.O.2
⇨ L.O.3
❑ L.O.4

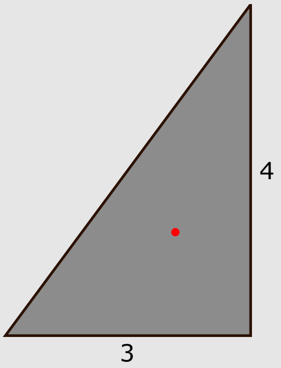
Then define `y_1` and `y_2`, as well as the value of Δy: `dy`




```
x = @(y) 3 - 3/4 * y;
y_1 = 0; y_2 = 4; dy = 0.1;
% start, end, and step size
.
.
.
```

3 – Numerical Implementation

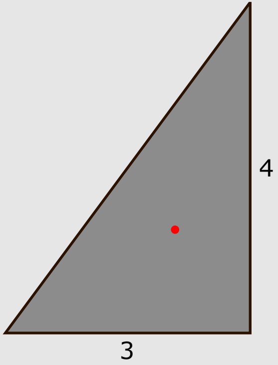
Then we can define the array of y-locations to evaluate:




```
x = @(y) 3 - 3/4 * y;
y_1 = 0; y_2 = 4; dy = 0.1;
y_locations = y_1 : dy : y_2;
% colon notation:
%   <start> : <step size> : <end>
.
```

3 – Numerical Implementation

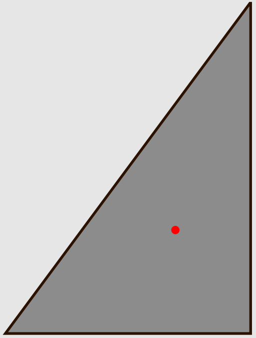
The “height” of each rectangle is then evaluated with **x(y)**:




```
x = @(y) 3 - 3/4 * y;
y_1 = 0; y_2 = 4; dy = 0.1;
y_locations = y_1 : dy : y_2;
x_values = x(y_locations);
% applying the x(y) function
.
```

3 – Numerical Implementation

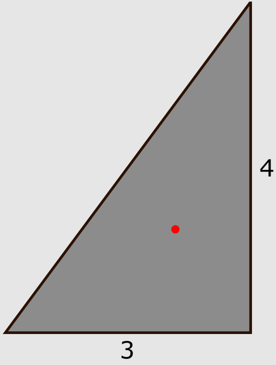
Evaluate the area of each rectangle as $h \times w$:




```
x = @(y) 3 - 3/4 * y;
y_1 = 0; y_2 = 4; dy = 0.1;
y_locations = y_1 : dy : y_2;
x_values = x(y_locations);
delta_A_values = x_values .* dy;
% calculating the dAs
```

3 – Numerical Implementation

And finally, evaluate the total area:




```
x = @(y) 3 - 3/4 * y;
y_1 = 0; y_2 = 4; dy = 0.1;
y_locations = y_1 : dy : y_2;
x_values = x(y_locations);
delta_A_values = x_values .* dy;
A = sum(delta_A_values)            % total area
```

```
A =
     6.1500   % analytic answer: 6
```

Command Window Output

3 – Numerical Integration — Numerator

Now let's evaluate the numerator of the centroid calculation:

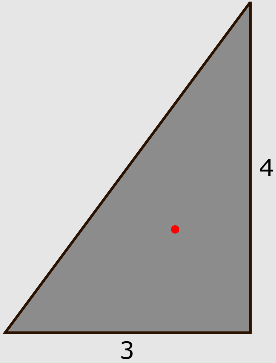
$$\int_{y_1}^{y_2} \left[x_{C_{dA}}(y)\right] dA = \int_{y_1}^{y_2} \underbrace{\left[\frac{x(y)}{2}\right]}_{x_{C_{dA}}} \underbrace{\left[x(y)\right] dy}_{dA}$$

$$\int_{y_1}^{y_2} \left[x_{C_{dA}}(y)\right] dA \approx \sum_{i=1}^{N} \underbrace{\left[\frac{x(y_i)}{2}\right]}_{x_{C_{dA_i}}} \underbrace{\left[x\left(y_i\right)\right] \Delta y}_{\Delta A_i}$$

3 – Numerical Implementation

✓ L.O.1
✓ L.O.2
➩ L.O.3
❏ L.O.4

Let’s add one line to the end of our previous code:



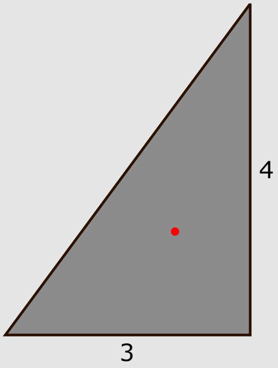

```
x = @(y) 3 - 3/4 * y;
y_1 = 0; y_2 = 4; dy = 0.1;
y_locations = y_1 : dy : y_2;
x_values = x(y_locations);
delta_A_values = x_values .* dy;
A = sum(delta_A_values)
numerator = sum(x_values./2 .* delta_A_values)
```

```
numerator = 6.2269
```

Command Window Output

3 – Full Centroid Calculation

To evaluate the centroid, we divide **numerator/A**:



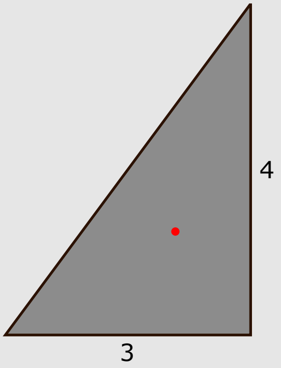

```
x = @(y) 3 - 3/4 * y;
y_1 = 0; y_2 = 4; dy = 0.1;
y_locations = y_1 : dy : y_2;
x_values = x(y_locations);
delta_A_values = x_values .* dy;
A = sum(delta_A_values)
numerator = sum(x_values./2 .* delta_A_values)
x_C = numerator / A
```

```
x_C = 1.0125
```

Command Window Output

3 – Full Centroid Calculation

Note: this value is taken from the "base", which is at **x=3**:




```
x = @(y) 3 - 3/4 * y;
y_1 = 0; y_2 = 4; dy = 0.1;
y_locations = y_1 : dy : y_2;
x_values = x(y_locations);
delta_A_values = x_values .* dy;
A = sum(delta_A_values)
numerator = sum(x_values./2 .* delta_A_values)
x_C = 3 - numerator / A
```

```
x_C = 1.9875   % analytic answer: 2
```

Command Window Output

4 – Acknowledging the Inaccuracy

But wait ... we know `A = 6` and `x_C = 2`.

Why is the code saying `A = 6.15` and `x_C = 1.9875`?

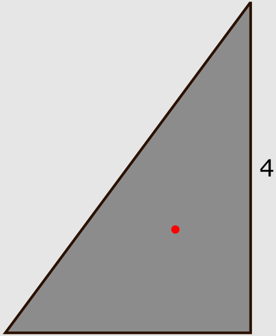
Numerical integration is, by *definition*, an ***approximation***.

Recall: $\int_{y_1}^{y_2} x(y)dy = \lim_{\Delta y \to 0} \left[\sum_{i=1}^{N} x(y_i)\Delta y \right]$

4 – Addressing the Inaccuracy

✓ L.O.1
✓ L.O.2
✓ L.O.3
➪ L.O.4

Let's return to our example. Set `dy = 0.01`:




```
x = @(y) 3 - 3/4 * y;
y_1 = 0; y_2 = 4; dy = 0.01; % smaller dy
y_locations = y_1 : dy : y_2;
x_values = x(y_locations);
delta_A_values = x_values .* dy;
A = sum(delta_A_values)
numerator = sum(x_values./2 .* delta_A_values);
x_C = 3 - numerator / A
```

```
A = 6.0150
```

Command Window Output

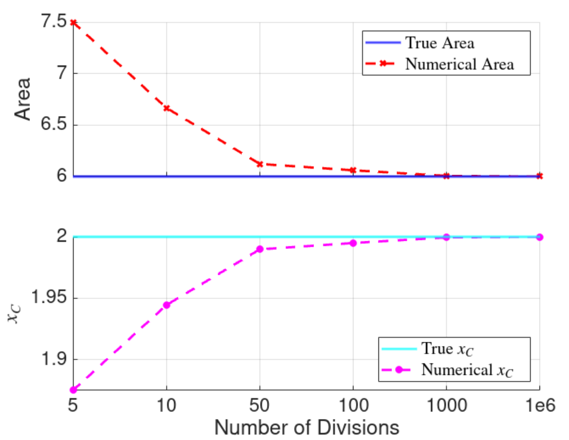
```
x_C = 1.9988
```

Command Window Output

4 – Convergence

To extrapolate this, we look at how the values for **`A`** and **`x_C`** changes with **`dy`**.

By varying the number of divisions, the values of each approximation converge towards their analytic "true" values.



4 – Providing Convergence Code

The code that was used to produce the convergence graph on the prior slide will not be reviewed in this lecture video.

But for those who are curious, it is provided here on the next slide in the downloaded version.

There are a number of things that we haven't covered yet. Feel free to look them up!

4 – Convergence Code: Part 1


```
test_divisions = [5, 10, 50, 1e2, 1e3, 1e6];     % different N
areas = zeros(1, length(test_divisions));        % allocate for areas
x_centroids = zeros(1, length(test_divisions)); % allocate for centroids
x = @(y) 3 - 3/4 * y;                            % anonymous function
for i = 1 : length(test_divisions)               % loop over N
  num_points = test_divisions(i);
  y_points = linspace(0,4,num_points);           % y values array
  dy = y_points(2) - y_points(1);                % delta y
  x_values = x(y_points);                        % x "heights"
  areas(i) = sum(x_values .* dy);                % area of triangle
  % x centroid value (below)
  x_centroids(i) = 3 - sum((x_values./2) .* (x_values.*dy)) ./ areas(i);
end
```

4 – Convergence Code: Part 2


```
figure                                                   % create figure
subplot(2,1,1);                                          % 2x1 subplot (plot 1)
yline(6, '-b', 'LineWidth', 2)                           % horizontal line - A
hold on                                                  % preserve graph
plot(areas, 'x--r', 'LineWidth', 2)                      % plot area values
xticks(1:6)                                              % make axis ticks
xticklabels([])                                          % remove tick labels
ylim([6,7.5])                                            % set y limits
ylabel('Area')                                           % label y axis
legend('True Area', 'Numerical Area', 'Interpreter', 'latex', ...
  'Location', 'northeast')
set(gca, 'FontSize', 16)                                 % increase font size
grid on                                                  % turn on grid lines
```

4 – Convergence Code: Part 3

✓ L.O.1
✓ L.O.2
✓ L.O.3
⇨ L.O.4

```
subplot(2,1,2);                                    % 2x1 subplot (plot 2)
yline(2, '-c', 'LineWidth', 2)                     % horizontal line - x_C
hold on                                            % preserve graph
plot(x_centroids, '*--m', 'LineWidth', 2)          % plot x centroids
xticks(1:6)                                        % make axis ticks
xticklabels({'5', '10', '50', '1e2', '1e3', '1e6'}) % label axis ticks
ylim([1.875, 2])                                   % set y limits
xlabel('Number of Divisions')                      % label x axis
ylabel('$x_{C}$', 'Interpreter', 'latex')          % label y axis
legend('True $x_{C}$', 'Numerical $x_{C}$', 'Interpreter', 'latex', ...
  'Location', 'southeast')
set(gca, 'FontSize', 16)                           % increase font size
grid on                                            % turn on grid lines
```

5 – Summary

✓ L.O.1
✓ L.O.2
✓ L.O.3
✓ L.O.4

This lecture covered:

✓ What numeric integration is

An approach to evaluating integrals whereby calculus is simplified down to algebraic summations.

✓ Why numeric integration is important on computers

Unlike humans, computers cant just *do* an integral.But computers are really good at algebra, so numerics play an important role in engineering computation.

5 – Summary

✓ L.O.1
✓ L.O.2
✓ L.O.3
✓ L.O.4

✓ Applying numeric integration to centroid calculations

By first discretizing the domain, then evaluating the function, and adding the areas of all the boxes, the numeric approach approximates a true integral.

And its accuracy improves as the domain discretization is done with more divisions.

✓ How numerical results converge

Changing the number of divisions in the domain improves accuracy by more closely approximating the true integral.